ERNIE: a tool to study the Tor network
– User's Guide –

by Karsten Loesing <karsten@torproject.org>

August 19, 2010

1 Overview

Welcome to ERNIE! ERNIE is a tool to study the Tor network. ERNIE has been designed to process all kinds of data about the Tor network and visualize them or prepare them for further analysis. ERNIE is also the software behind the Tor Metrics Portal `http://metrics.torproject.org/`.

The acronym ERNIE stands for the *Enhanced R-based tor Network Intelligence Engine* (sorry for misspelling Tor). Why ERNIE? Because nobody liked BIRT (Business Intelligence and Reporting Tools) that we used for visualizing statistics about the Tor network before writing our own software. By the way, reasons were that BIRT made certain people's browsers crash and requires JavaScript that most Tor user have turned off.

If you want to learn more about the Tor network, regardless of whether you want to present your findings on a website (like ERNIE does) or include them in your next Tor paper, this user's guide is for you!

2 Installation instructions

ERNIE depends on various other software tools. ERNIE is developed in a *Git* repository which is currently the only way to download it. ERNIE uses *Java* for parsing data, *R* for plotting graphs, and *PostgreSQL* for importing data into a database. Which of these tools you need depends on what tasks you are planning to use ERNIE for. In most cases it is not required to install all these tools. For this tutorial, we assume Debian GNU/Linux 5.0 as operating system. Installation instructions for other platforms may vary.

2.1 Git 1.5.6.5

Currently, the only way to download ERNIE is to clone its Git branch.

Install Git 1.5.6.5 (or higher) and check that it's working:

```
$ sudo apt-get install git-core
$ git --version
```

2.2 Java 6

ERNIE requires Java to parse data from various data sources and write them to one or more data sinks. Java is required for most use cases of ERNIE.

Add the non-free repository to the apt sources in `/etc/apt/sources.list` by changing the line (mirror URL may vary):

```
deb http://ftp.ca.debian.org/debian/ lenny main
```

to

```
deb http://ftp.ca.debian.org/debian/ lenny main non-free
```

Fetch the package list, install Sun Java 6, and set it as system default:

```
$ sudo apt-get update
$ sudo apt-get install sun-java6-jdk
$ sudo update-alternatives --set java
      /usr/lib/jvm/java-6-sun/jre/bin/java
$ sudo update-alternatives --set javac
      /usr/lib/jvm/java-6-sun/bin/javac
```

Check that Java 6 is installed and selected as default:

```
$ java -version
$ javac -version
```

2.3 Ant 1.7

ERNIE comes with an Ant build file that facilitates common build tasks. If you want to use Ant to build and run ERNIE, install Ant and check its installed version (tested with 1.7):

```
$ sudo apt-get install ant
$ ant -version
```

2.4 R 2.8 and ggplot2

ERNIE uses R and the R library *ggplot2* to visualize anaylsis results for presentation on a website or for inclusion in publications. ggplot2 requires at least R version 2.8 to be installed.

Add a new line to `/etc/apt/sources.list`:

```
deb http://cran.cnr.berkeley.edu/bin/linux/debian lenny-cran/
```

Download the package maintainer's public key ("Johannes Ranke (CRAN Debian archive) <jranke@uni-bremen.de>"):

```
$ gpg --keyserver pgpkeys.pca.dfn.de --recv-key 381BA480
$ gpg --export 381BA480 | sudo apt-key add -
```

Install the most recent R version:

```
$ sudo apt-get update
$ sudo apt-get -t unstable install r-base
```

Start R to check its version (must be 2.8 or higher) and install ggplot2. Do this as root, so that the installed package is available to all system users:

```
$ sudo R
> install.packages("ggplot2")
> q()
```

Confirm that R and ggplot2 are installed:

```
$ R
> library(ggplot2)
> q()
```

2.5 PostgreSQL 8.3

ERNIE uses PostgreSQL to import data into a database for later analysis. This feature is not required for most use cases of ERNIE, but only for people who prefer having the network data in a database to execute custom queries.

Install PostgreSQL 8.3 using apt-get:

```
$ sudo apt-get install postgresql-8.3
```

Create a new database user `ernie` to insert data and run queries. This command is executed as unix user `postgres` and therefore as database superuser `postgres` via ident authentication. The `-P` flag issues a password prompt for the new user. There is no need to give the new user superuser privileges or allow it to create databases or new roles.

```
$ sudo -u postgres createuser -P ernie
```

Create a new database schema `tordir` owned by user `ernie` (using option `-O`). Again, this command is executed as `postgres` system user to make use of ident authentication.

```
$ sudo -u postgres createdb -O ernie tordir
```

Log into the database schema as user `ernie` to check that it's working. This time, ident authentication is not available, since there is no system user `ernie`. Instead, we will use password authentication via a TCP connection to localhost (using option `-h`) as database user `ernie` (using option `-U`).

```
$ psql -h localhost -U ernie tordir
tordir=> \q
```

2.6 ERNIE

Finally, you can install ERNIE by cloning its Git branch:

```
$ git clone git://git.torproject.org/ernie
```

This command should create a directory `ernie/` which we will consider the working directory of ERNIE.

3 Getting started with ERNIE

The ERNIE project was started as a simple tool to parse Tor relay descriptors and plot graphs on Tor network usage for a website. Since then, ERNIE has grown to a tool that can process all kinds of Tor network data for various purposes, including but not limited to visualization.

We think that the easiest way to get started with ERNIE is to walk through typical use cases in a tutorial style and explain what is required to set up ERNIE. These use cases have been chosen from what we think are typical applications of ERNIE.

3.1 Visualizing network statistics

Write me.

3.2 Importing relay descriptors into a database

As of February 2010, the relays and directories in the Tor network generate more than 1 GB of descriptors every month. There are two approaches to process these amounts of data: extract only the relevant data for the analysis and write them to files, or import all data to a database and run queries on the database. ERNIE currently takes the file-based approach for the Metrics Portal, which works great for standardized analyses. But the more flexible way to research the Tor network is to work with a database.

This tutorial describes how to import relay descriptors into a database and run a few example queries. Note that the presented database schema is limited to answering basic questions about the Tor network. In order to answer more complex questions, one would have to extend the database schema and Java classes which is sketched at the end of this tutorial.

3.2.1 Preparing database for data import

The first step in importing relay descriptors into a database is to install a database management system. See Section 2.5 for installation instructions of PostgreSQL 8.3 on Debian GNU/Linux 5.0. Note that in theory, any other relational database that has a working JDBC 4 driver should work, too, possibly with minor modifications to ERNIE.

Import the database schema from file `tordir.sql` containing two tables that we need for importing relay descriptors plus two indexes to accelerate queries. Check that tables have been created using `\dt`. You should see a list containing the two tables `descriptor` and `statusentry`.

```
$ psql -h localhost -U ernie -f tordir.sql tordir
$ psql -h localhost -U ernie tordir
tordir=> \dt
tordir=> \q
```

A row in the `statusentry` table contains the information that a given relay (that has published the server descriptor with ID `descriptor`) was contained in the network status consensus published at time `validafter`. These two fields uniquely identify a row in the `statusentry` table. The other fields contain boolean values for the flags that the directory authorities assigned to the relay in this consensus, e.g., the Exit flag in `isexit`. Note that for the 24 network status consensuses of a given day with each of them containing 2000 relays, there will be 24×2000 rows in the `statusentry` table.

The `descriptor` table contains some portion of the information that a relay includes in its server descriptor. Descriptors are identified by the `descriptor` field which corresponds to the `descriptor` field in the `statusentry` table. The other fields contain further data of the server descriptor that might be relevant for analyses, e.g., the platform line with the Tor software version and operating system of the relay.

Obviously, this data schema doesn't match everyone's needs. See the instructions below for extending ERNIE to import other data into the database.

3.2.2 Downloading relay descriptors from the metrics website

In the next step you will probably want to download relay descriptors from the metrics website `http://metrics.torproject.org/data.html#relaydesc`. Download the `v3 consensuses` and/or `server descriptors` of the months you want to analyze. The server descriptors are the documents that relays publish at least every 18 hours describing their capabilities, whereas the v3 consensuses are views of the directory authorities on the available relays at a given time. For this tutorial you need both v3 consensuses and server descriptors. You might want to start with a single month of data, experiment with it, and import more data later on. Extract the tarballs to a new directory `archives/` in the ERNIE working directory.

3.2.3 Configuring ERNIE to import relay descriptors into a database

ERNIE can be used to read data from one or more data sources and write them to one or more data sinks. You need to configure ERNIE so that it knows to use the downloaded relay descriptors as data source and the database as data sink. Add the following two lines to your `config` file:

```
ImportDirectoryArchives 1
WriteRelayDescriptorDatabase 1
```

You further need to provide the JDBC string that ERNIE shall use to access the database schema `tordir` that we created above. The config option with the JDBC string for a local PostgreSQL database might be (without line break):

```
RelayDescriptorDatabaseJDBC
  jdbc:postgresql://localhost/tordir?user=ernie&password=password
```

3.2.4 Importing relay descriptors using ERNIE

Now you are ready to actually import relay descriptors using ERNIE. Create a directory for Java class files, compile the Java source files, and run ERNIE. All these steps are performed by the default target in the provided Ant task.

```
$ ant
```

Note that the import process might take between a few minutes and an hour, depending on your hardware. You will notice that ERNIE doesn't write progress messages to the standard output, which is useful for unattended installations with only warnings being mailed out by cron. You can change this behavior and make messages on the standard output more verbose by setting `java.util.logging.ConsoleHandler.level` in `logging.properties` to `INFO` or `FINE`. Alternately, you can look at the log file `log.0` that is created by ERNIE.

If ERNIE finishes after a few seconds, you have probably put the relay descriptors at the wrong place. Make sure that you extract the relay descriptors to sub directories of `archives/` in the ERNIE working directory.

If you interrupt ERNIE, or if ERNIE terminates uncleanly for some reason, you will have problems starting it the next time. ERNIE uses a local lock file called `lock` to make sure that only a single instance of ERNIE is running at a time. If you are sure that the last ERNIE instance isn't running anymore, you can delete the lock file and start ERNIE again.

If all goes well, you should now have the relay descriptors of 1 month in your database.

3.2.5 Import optimizations

If you are importing a very large amount of descriptors into the database at once, it is possible to speed up the loading process by the order of a magnitude or more. I would recommend following this section if you are importing a year or more of data, or if importing descriptors is causing unnecessarily high load on your machine. Certain sacrifices concerning integrity of data will have to be made, so be careful of this. Furthermore, the following steps should only be used if you are starting from scratch (unless you want a big mess to clean up!).

Removing the indexes and primary key constraints and re-building them after can benefit performance.

```
ALTER TABLE descriptor
DROP CONSTRAINT descriptor_pkey;

ALTER TABLE descriptor
DROP CONSTRAINT descriptor_pkey;

DROP INDEX descriptorid;

DROP INDEX statusentryid;
```

Secondly, you may want to comment out a few lines in the source file `src/org/torproject/metrics/db/RelayDescriptorDatabaseImporter.java` since they involve integrity checking, and re-build it with ant when you are done (`ant compile`). The following blocks in this file can be commented out with no effect (in the addDescriptor() and addStatusentry() methods). TODO - make a config option for this.

```
    this.psDs.setString(1, descriptor);
    ResultSet rs = psDs.executeQuery();
    rs.next();
    if (rs.getInt(1) > 0) {
      return;
    }


    this.psRs.setTimestamp(1, validAfterTimestamp, cal);
    this.psRs.setString(2, descriptor);
    ResultSet rs = psRs.executeQuery();
    rs.next();
    if (rs.getInt(1) > 0) {
      return;
    }
```

Another way to speed up imports greatly is to turn off fsync. This, however, will require editing the Postgres config and restarting the server. Be careful with this option, because it is possible corrupt your data in case of a hard disk or power failure, so make sure you really trust your hardware! Otherwise, it is the best way to enhance performance. To turn fsync off, add the line `fsync=off` to `Postgresql.conf` (found in `/etc` or `/usr/local/pgsql/data`) and restart the server. A similar, but safer way of achieving this effect can be done by increasing `wal_buffers`. Other Postgres config options to improve performance are `shared_buffers`, `work_mem`, and `archive_mode`. Please, read the Postgres docs for more information on these before changing anything.

(`http://www.postgresql.org/docs/8.4/static/runtime-config-resource.html`)

Now, run ant (See section 3.2.4) as usual. After importing data (which still may take a few hours depending on your hardware and config), you will have to re-build the indexes and constraints. This means that duplicate rows have to be deleted. Luckily, there shouldn't be too many duplicate rows to begin with, but it does happen. First, re-build the indexes:

```
CREATE INDEX descriptorid ON descriptor
    USING btree (descriptor);

CREATE INDEX statusentryid ON statusentry
    USING btree (descriptor, validafter);
```

Then, re-build the primary keys:

```
ALTER TABLE ONLY descriptor
    ADD CONSTRAINT descriptor_pkey PRIMARY KEY (descriptor);

ALTER TABLE ONLY statusentry
    ADD CONSTRAINT statusentry_pkey
        PRIMARY KEY (validafter, descriptor);
```

Re-building the primary keys may fail if there are duplicates in the descriptor table. Here is an example of how to delete the duplicate rows in the descriptor table, using Postgres' built-in hidden ctid field. A similar methodology can be used for the statusentry table:

```
CREATE TABLE descriptor_dupes AS
SELECT descriptor
FROM descriptor
GROUP BY descriptor
HAVING COUNT(*) > 1;

DELETE FROM descriptor WHERE ctid IN (
    SELECT MAX(ctid) FROM descriptor
    WHERE descriptor IN (SELECT descriptor FROM descriptor_dupes)
    GROUP by descriptor)

DROP TABLE descriptor_dupes;
```

This will remove the first level of duplicates. There may be more, in which case the query can be run again until there are none left. Anyway, after deleting the duplicates, the primary keys can be re-built again.

You can experiment with the config option AutoCommitCount, but you will see diminishing turns with values above 500. This tells Postgres how many INSERTS to do per transaction, since it will write to the disk only every 500 inserts.

You are all set to go! Remember to reset the Postgres config options and turn fsync back on if you turned it back off. Wow, inserting data is so much faster!

3.2.6 Example queries

In this tutorial, we want to give you a few examples for using the database schema with the imported relay descriptors to extract some useful statistics about the Tor network.

In the first example we want to find out how many relays have been running on average per day and how many of these relays were exit relays. We only need the `statusentry` table for this evaluation, because the information we are interested in is contained in the network status consensuses.

The SQL statement that we need for this evaluation consists of two parts: First, we find out how many network status consensuses have been published on any given day. Second, we count all relays and those with the Exit flag and divide these numbers by the number of network status consensuses per day.

```
$ psql -h localhost -U ernie tordir
tordir=> SELECT DATE(validafter),
    COUNT(*) / relay_statuses_per_day.count AS avg_running,
    SUM(CASE WHEN isexit IS TRUE THEN 1 ELSE 0 END) /
      relay_statuses_per_day.count AS avg_exit
  FROM statusentry,
    (SELECT COUNT(*) AS count, DATE(validafter) AS date
      FROM (SELECT DISTINCT validafter FROM statusentry)
      distinct_consensuses
      GROUP BY DATE(validafter)) relay_statuses_per_day
  WHERE DATE(validafter) = relay_statuses_per_day.date
  GROUP BY DATE(validafter), relay_statuses_per_day.count
  ORDER BY DATE(validafter);
tordir=> \q
```

Executing this query should finish within a few seconds to one minute, again depending on your hardware. The result might start like this (truncated here):

```
    date    | avg_running | avg_exit
------------+-------------+----------
 2010-02-01 |        1583 |      627
 2010-02-02 |        1596 |      638
 2010-02-03 |        1600 |      654
:
```

In the second example we want to find out what Tor software versions the relays have been running. More precisely, we want to know how many relays have been running what Tor versions on micro version granularity (e.g., 0.2.2) on average per day?

We need to combine network status consensuses with server descriptors to find out this information, because the version information is not contained in the consensuses (or at least, it's optional to be contained in there; and after all, this is just an example). Note that we cannot focus on server descriptors only and leave out the consensuses for this analysis, because we want our analysis to be limited to running relays as confirmed by the directory authorities and not include all descriptors that happened to be published at a given day.

The SQL statement again determines the number of consensuses per day in a sub query. In the next step, we join the `statusentry` table with the `descriptor` table for all rows contained in the `statusentry` table. The left join means that we include `statusentry` rows even if we do not have corresponding lines in the `descriptor` table. We determine the version by skipping the first 4 characters of the platform string that should contain `"Tor "` (without quotes) and cutting off after another 5 characters. Obviously, this approach is prone to errors if the platform line format changes, but it should be sufficient for this example.

```
$ psql -h localhost -U ernie tordir
tordir=> SELECT DATE(validafter) AS date,
    SUBSTRING(platform, 5, 5) AS version,
    COUNT(*) / relay_statuses_per_day.count AS count
  FROM
    (SELECT COUNT(*) AS count, DATE(validafter) AS date
    FROM (SELECT DISTINCT validafter
      FROM statusentry) distinct_consensuses
    GROUP BY DATE(validafter)) relay_statuses_per_day
  JOIN statusentry
    ON relay_statuses_per_day.date = DATE(validafter)
  LEFT JOIN descriptor
    ON statusentry.descriptor = descriptor.descriptor
  GROUP BY DATE(validafter), SUBSTRING(platform, 5, 5),
    relay_statuses_per_day.count, relay_statuses_per_day.date
  ORDER BY DATE(validafter), SUBSTRING(platform, 5, 5);
tordir=> \q
```

Running this query takes longer than the first query, which can be a few minutes to half an hour. The main reason is that joining the two tables is an expensive database operation. If you plan to perform many evaluations like this one, you might want to create a third table that holds the results of joining the two tables of this tutorial. Creating such a table to speed up queries is not specific to ERNIE and beyond the scope of this tutorial.

The (truncated) result of the query might look like this:

```
    date    | version | count
------------+---------+-------
 2010-02-01 | 0.1.2   |    10
 2010-02-01 | 0.2.0   |   217
 2010-02-01 | 0.2.1    |   774
 2010-02-01 | 0.2.2    |    75
 2010-02-01 |          |   505
 2010-02-02 | 0.1.2    |    14
 2010-02-02 | 0.2.0    |   328
 2010-02-02 | 0.2.1    |  1143
 2010-02-02 | 0.2.2    |   110
:
```

Note that, in the fifth line, we are missing the server descriptors of 505 relays contained in network status consensuses published on 2010-02-01. If you want to avoid such missing values, you'll have to import the server descriptors of the previous month, too.

3.2.7 Extending ERNIE to import further data into the database

In this tutorial we have explained how to prepare a database, download relay descriptors, configure ERNIE, import the descriptors, and execute example queries. This description is limited to a few examples by the very nature of a tutorial. If you want to extend ERNIE to import further data into your database, you will have to perform at least two steps: extend the database schema and modify the Java classes used for parsing.

The first step, extending the database schema, is not specific to ERNIE. Just add the fields and tables to the schema definition.

The second step, modifying the Java classes used for parsing, is of course specific to ERNIE. You will have to look at two classes in particular: The first class, `RelayDescriptorDatabaseImporter`, contains the prepared statements and methods used to add network status consensus entries and server descriptors to the database. The second class, `RelayDescriptorParser`, contains the parsing logic for the relay descriptors and decides what information to add to the database, among other things.

This ends the tutorial on importing relay descriptors into a database. Happy researching!

3.3 Aggregating relay and bridge descriptors

Write me.

4 Software architecture

Write me. In particular, include overview of components:

- *Data sources and data sinks*
- *Java classes with data sources and data sinks*
- *R scripts to process CSV output*

- *Website*

5 Tor Metrics Portal setup

Write me. In particular, include documentation of deployed ERNIE that runs the metrics website. This documentation has two purposes: First, a reference setup can help others creating their own ERNIE configuration that goes beyond the use cases as described above. Second, we need to remember how things are configured anyway, so we can as well document them here.